UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Automated Real-Time Performance Feedback and Time Study using Computer Vision Technology

Hagen Alexander von Petersdorff

15355802

Final year project presented in partial fulfilment to the requirements for the degree of Bachelors of Industrial Engineering at Stellenbosch University

Study Leader: Dr A van der Merwe

October 2011

ABSTRACT

This final year project report describes the creation of a computer vision solution to assess worker performance in a manual labour environment. Time-study data is gathered by the system and performance feedback is given to workers in real-time. The system consists of a Microsoft Xbox Kinect camera, connected via USB to a PC. The Kinect camera is used alongside OpenNI software to perform skeletal-tracking on the worker, and this data is processed by an application created in C++ to perform cycle recognition. Performance is calculated by assessing the time period between successive cycles and simple feedback is given to the worker after each cycle by an LCD display. Performance data are stored for subsequent analysis.

DECLARATION

I, the undersigned, hereby declare that the work contained in this thesis final year project is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

…….. ……

Date

ECSA EXIT LEVEL OUTCOMES REFERENCES

The following table include references to sections in this report where ECSA exit level outcomes are addressed.

Exit Level Outcome	**Section(s)**	**Pages(s)**
1. Problem Solving	1. 3.	1-6 10-37
5. Engineering methods, skills & tools, incl. IT	2.2. 3. 4.	6-8 10-37 37-41
6. Professional & technical communication	Entire document	
9. Independent learning ability	2. 3. 4.	6-10 10-37 37-41
10. Engineering professionalism	3.1. 3.2. 4.2.	10-16 16-22 37-41

Explanation on satisfaction of ECSA Exit Level Outcomes:

1. Problem solving is evident in of the definition of the problem. It is also handled in the idea conception and formulation of a solution. From the literature study problem solving was used to make a decision regarding the method that was best suited for automation. In the development of the system problems were identified and eliminated leading to the final generation of a solution to the original problem.
5. Engineering methods, skills & tools were used in the process of understanding the available methodologies to evaluate labour performance. The work sampling methodology was then coded creating the program used to solve the problem.
6. Professional and technical communication is demonstrated in this final report as well as the final presentation at the oral examination.
9. This project displays the independent learning with successful identification of the literature required to find a solution to an engineering problem and the application of this knowledge to develop a system for further use. Further insights towards the improvement of the program are given in the project conclusion.
10. The tools available to determine the labour performance were examined in the literature review of this project. A number of decisions were made in the development of the system.

SUMMARY

This final year project details the innovative design, development, implementation and evaluation of a novel method to perform data gathering for time-studies, and an innovative proposal to provide workers in manual labour environment with constant performance feedback.

The primary aim of the system is to perform joint tracking on a worker, observe and record work cycles, record this data for analysis, and to provide simple and intuitive performance feedback to the worker performing the task. The system aims to be as non-intrusive and robust as possible. Calculations and algorithms are performed as simply as possible to enable the analysis to execute in real-time.

A Microsoft Kinect camera is used, which uses structured infrared light to create a dense digital 3D representation of a scene. The camera is connected to a computer which runs a standalone executable application, coded in C++ by the author.

Joint tracking is performed by OpenNI software, and the data gathered from the joints represented by the worker's hands are parsed over a period of time to automatically find significant nodes in the scene. The system then records nodes visits and performs a pattern detection algorithm to extract the work cycle being performed. The system detects and records these cycles if they are performed subsequently, and calculates the time between cycles. Worker performance is calculated and feedback is given to the worker via a simple colour scale after each completed cycle. This data is gathered in a log file for later analysis.

OPSOMMING

Hierdie finale jaar projek die gee die besonderhede van die innoverende ontwerp, ontwikkeling, implementering en evaluering van' n nuwe metode om data-insameling vir tyd-studies uit te voer, en 'n innoverende voorstel om werkers in die handearbeid omgewing te voorsien met konstante prestasie terugvoer.

Die primêre doel van die stelsel is om skelet-tracking van 'n werknemer uit te voer, werk siklusse waar te neem en vas te lê, en die data aan te teken vir analise. Dit gee ook eenvoudig en intuïtief prestasie terugvoer aan die werker wat die taak verrig. Die stelsel is daarop gemik om nie-indringend en so sterk as moontlik te wees. Berekeninge en algoritmes is so eenvoudig as moontlik uitgevoer sodat die analise in werklike tyd uitgevoer kan word.

'N Microsoft Xbox Kinect kamera word gebruik, wat gebruik maak van gestruktureerde infrarooi lig om ‘n digte digitale 3D voorstelling van 'n toneel te skep. Die kamera is verbind tot 'n rekenaar wat ‘n selfstandige uitvoerbare program, gekodeer in C + + deur die skrywer, uitvoer.

Skelet-tracking is uitgevoer deur OpenNI sagteware, en die data wat ingesamel is van die gewrigte wat verteenwoordig word deur die gebruiker se hande is geparseerd oor 'n beraamde tydperk, sodat belangrike nodusse outomaties in die toneel gevind kan word. Die stelsel neem die besoek van nodes op en voer 'n patroon opsporing algoritme om die werk siklus wat uitgevoer is te onttrek. Die stelsel bespeur en neem hierdie siklusse op as hulle daarna uitgevoer is, en bereken die tyd tussen afgelope siklusse. Werker prestasie is bereken en terugvoering word gegee aan die werker deur 'n eenvoudige kleur skaal na elke voltooide siklus. Hierdie data is versamel in 'n log-lêer vir latere analise.

TABLE OF CONTENTS

ABSTRACT ..... **I**

DECLARATION ..... **II**

ECSA EXIT LEVEL OUTCOMES REFERENCES ..... **III**

SUMMARY ..... **V**

OPSOMMING ..... **VI**

TABLE OF CONTENTS ..... **VII**

LIST OF FIGURES ..... **IX**

1. INTRODUCTION **1**

1.1. BACKGROUND 1

1.2. RESEARCH QUESTION 1

1.2.1. Goals of System *2*

1.3. DELIMITATIONS 3

1.4. DEFINITION OF TERMS 4

1.4.1. The System *4*

1.4.2. Effective *4*

1.5. ROADMAP OF THIS DOCUMENT 4

1.6. SUMMARY 5

2. LITERATURE RESEARCH **6**

2.1. PERFORMANCE FEEDBACK IN MANUAL LABOUR ENVIRONMENT 6

2.2. TIME-STUDY 6

2.2.1. Performing time studies *6*

2.3. JOINT-TRACKING AND POSE ESTIMATION USING COMPUTER VISION 8

3. REAL-TIME PERFORMANCE FEEDBACK AND TIME STUDY USING COMPUTER VISION TECHNOLOGY **10**

3.1. CONSIDERATION OF ALTERNATIVE METHODS 10

3.1.1. Computer vision: Hardware *10*

3.1.2. Computer Vision: Software *12*

3.1.3. Cycle Recognition *13*

3.2. CHOOSING ALTERNATIVES AND PLANNING METHODOLOGY 16

3.2.1. Hardware: Microsoft Xbox Kinect Camera *17*

3.2.2. Software: OpenNI SDK with NITE Middleware *18*

3.2.3. Cycle Recognition Algorithm *19*
3.2.4. Summary of Cycle Recognition *21*
3.2.5. Performance Calculation and Feedback Method *21*
3.3. EXECUTION METHODOLOGY 22
3.3.1. Computer Vision System *23*
3.3.2. Skeletal Tracking *26*
3.3.3. Joint Coordinate Feed *27*
3.3.4. Training *27*
3.3.5. Recognition *33*
3.3.6. Performance Calculation *34*
3.3.7. Feedback to Worker *35*
3.3.8. Data Logging *36*
4. DEMONSTRATION OF METHOD 37
4.1. AIM 37
4.2. METHOD 37
4.2.1. Setup *37*
4.2.2. Execution *38*
4.2.3. Data *39*
4.2.4. Results *39*
4.2.5. Discussion of Results *40*
5. CONCLUSION 42
6. FUTURE WORK 43
REFERENCES 45
APPENDIX A: OPENNI AND NITE INSTALLATION INSTRUCTIONS 47
APPENDIX B: DETAILED KINECT SPECIFICATIONS 48
APPENDIX C: FILE: *"CYCLERECOGNITION.CPP"* 49
APPENDIX D: FILE: "*STRINGFUNCTIONS.CPP*" 55
APPENDIX E: FILE: "*SKELETONFUNCTIONS.CPP*" 56

LIST OF FIGURES

FIGURE 1: DOCUMENT ROADMAP 4
FIGURE 2: THE KINECT'S PATTERN PROJECTION AS SEEN BY AN INFRARED CAMERA *(PHOTOGRAPH: A. PENVEN, SOME RIGHTS RESERVED)* 8
FIGURE 3: STRUCTURED LIGHT 3D SCANNING BY MEASURING DISTORTION IN PROJECTED STRIPE PATTERNS (SOURCE UNKNOWN) 9
FIGURE 4: AN ACTOR WEARS A MOTION CAPTURE SUIT WHICH ENABLES SKELETAL TRACKING FOR CHARACTER ANIMATION (SOURCE: INTERNET) 10
FIGURE 5: CYCLE RECOGNITION METHODOLOGY 14
FIGURE 6: STEPS IN EXECUTION OF CYCLE RECOGNITION PROJECT 22
FIGURE 7: THE MICROSOFT XBOX KINECT CAMERA (IFIXIT.COM) 23
FIGURE 8: INTERNAL PROCESSING LAYOUT OF THE KINECT (SOURCE: PRIMESENSE) 24
FIGURE 9: 3D SCENE AS VIEWED BY THE KINECT (IMAGE SOURCE UNKNOWN) 25
FIGURE 10: SCREEN CAPTURE OF OPENNI SCENE ANALYSER PERFORMING USER IDENTIFICATION 26
FIGURE 13: SKELETAL TRACKING SHOWING JOINTS TRACKED (SOURCE: PRIMESENSE) 27
FIGURE 14: METHODOLOGY OF TRAINING PROCESS 28
FIGURE 15: MOCK UP OF FIELD MESH USING SPREAD SHEET SOFTWARE 29
FIGURE 16: DISTRIBUTION OF VALUES FOR CALCULATING FIELD MESH 30
FIGURE 17: SCREEN CAPTURE OF NODE IDENTIFICATION PROCESS 31
FIGURE 18: SCREEN CAPTURE: TRAINING COMPLETE AND BIN LOCATIONS FIXED 32
FIGURE 19: SCREEN CAPTURE OF OUTPUT OF A PEFORMANCE FEEDBACK SCREEN (IMAGE BY AUTHOR) 35
FIGURE 20: COLOUR SCALE USED FOR PERFORMANCE FEEDBACK 35
FIGURE 21: TEST SETUP SHOWING WORKER AND ASSEMBLY BINS WITH LEGO BRICKS 38
FIGURE 22: COMPARISON OF OPERATOR AND CAMERA OBSERVATIONS 41

1. Introduction

This introductory section serves to provide the background of the project and to highlight the necessity of the project. This section also states the aims and objectives of this project and defines the boundaries of the project. Key terms used throughout the project are defined and a roadmap of this document is presented.

1.1. Background

South Africa is a country with a wealth of natural resources. Unfortunately though, South Africa's tertiary manufacturing sector is still underdeveloped and many of these abundant resources are exported to the East and processed there. Despite South Africa's apparent wealth, unemployment rates are high. Automation of manufacturing processes to increase productivity is detrimental to employment, and therefore it is vital that the effectiveness of South Africa's manual labour force is maximized in order to remain competitive.

While technology has improved drastically in recent years, methods for assessing labour performance are still supervisor intensive and have not developed much in concept since the introduction of time study methods over ninety years ago. With the wealth of affordable technology available today however, industrial engineers have an excellent opportunity to apply technology in innovative ways in order to improve labour productivity in the workplace.

1.2. Research Question

The author has recognized the potential benefits for automation of the time-study process, and has seen the opportunity to improve current industrial practices by providing constant performance feedback to workers in a manual labour environment, and by partially automating the time-study process. A system, which will be detailed in this document, will be implemented in order to address these opportunities.

Currently, a time-study requires that a supervisor observes one worker for a given time period, and manually times and notates this person's performance in order to assess what standard time can be used for the task. There are several problems with this method, such as: The cost

of employing the operator, subjective input of the operator, the worker's tendency to behave differently when being observed, and the time-period for which a worker is observed may not be long enough and cannot be extended due to cost. Thus, in order to improve the time-study process, the above points need to be addressed.

1.2.1. Goals of System

The goals of the system to be implemented can thus be summarized as follows:

i) To provide workers in a manual labour environment with constant performance feedback for certain repetitive tasks.

ii) To provide analysts with accurate, continuous time-study data.

iii) To partially automate the time-study process.

1.2.1.1. Constant Performance Feedback

In current wage incentive schemes, a workers performance is usually measured by the throughput of products produced by a worker (Freivalds, 2009). The problem with this incentive system is that there is a considerable time delay in providing feedback to the worker about his/her performance. A worker receiving performance feedback after a full days' or weeks' work is unlikely to be able to rectify his behaviour or claim inaccuracy, as there will be no data to support the workers claim. Data can then only be obtained the following day/week, which may be costly or disruptive, or have negative effects on the worker's morale. By enabling constant performance feedback to the worker, workers can immediately argue that a time-standard is incorrect, and there will not be any discrepancy when wage-incentives are paid, as the worker would have been able to observe his/her performance constantly during the task. The system may also provide psychological benefits to the work-dynamic, as workers will want to challenge themselves to see how fast they can go, and, being constantly observed, would not want to see their performance drop.

1.2.1.2. Time-Study Data for Analysis

Time study operators are usually forced to limit the number of measurements made, due to the time and cost involved, as described by Freivalds (Freivalds, 2009). By automating the time-study process, readings can be made over longer periods. This system would also allow tracking of a single user's output in situations where previously only a team-output and thus a team-incentive could be paid. The system could automatically detect pauses in work (as differentiated from slow work) so that uptime, fatigue allowances etc. can be determined and assessed automatically. The system's aim is thus to augment the analysis process of time-study, by automating the time-consuming data-gathering process.

1.2.1.3. Partial Automation of the Time-Study Process

A goal of this system is to augment the work of a time-study operator by observing a worker's actions over an extended period, thus aiding the operator in determining the standard time for that task. This will result in fewer tasks for the time-study operator, as there will be no need for the operator to spend extended periods observing a worker in order to calculate standard time, and will also result in greater accuracy.

A time-study operator is still required to control the process, and to provide additional factors such as a perceived performance rating or learning-curve type approximations on the task, as described in (Freivalds, 2009).

1.3. Delimitations

This project will focus on the gathering of data and will not attempt to analyse data or draw conclusions from them.

The system described created for this project has been designed to work in specific situations and environments and may not function correctly in a number of situations. Fixed bin locations and constant, repetitive work are two basic requirements for the system to function. The system is designed specifically for simple assembly tasks.

1.4. Definition of terms

Clarification of the following definitions is vital to understanding the underlying rationale of the document. Other terms, used in specific sections of this document, are defined in those respective sections.

1.4.1. The System

A system is defined as a group of interrelated and interdependent elements interacting with one another to achieve a common goal. "The system" as mentioned throughout this document, refers to all hardware, software elements which interact in order to fulfil this project's goals.

1.4.2. Effective

A number of different and contrasting definitions for the term "effective" are available. For the purposes of this document a system is referred to as effective if it is adequate to fulfil its purpose which in turn produces the desired or expected result.

1.5. Roadmap of This Document

A roadmap for this document is presented as Figure 1 below:

Figure 1: Document Roadmap

1.6. Summary

This introductory section served to provide the background and motivation for this project. Important terms were introduced and provided a roadmap of the remainder of the document.

2. Literature Research

The aim of this section is to investigate existing methods of providing workers in a manual labour environment with performance feedback. Existing methods of performing time-studies will also be examined, and the proposed technology to perform automated time studies will be introduced.

2.1. Performance Feedback in Manual Labour Environment

In current wage incentive schemes, a worker's performance is usually measured by the throughput of products produced by a worker (Freivalds, 2009). Providing meaningful feedback to workers in real-time is not well documented, as the technology to provide such feedback is a very recent development. There are various levels of feedback which are available to management however, which give feedback on a task after the completion of a number of tasks. Currently the most employed are:

- Post batch feedback
- Daily feedback
- Weekly feedback
- Post audit/Monthly feedback

2.2. Time-Study

Time study is a methodology used to determine the actual time required to complete a task. The calculated time is deemed to be a fair representation of the time required to complete a given task and takes allowances and delays into consideration. Time studies enable management to make effective decisions aimed at improving the efficiency of the entities operating within the system. It is important to realize that accurate time studies yield positive results and inaccurate time studies can create many problems (Freivalds, 2009).

2.2.1. Performing time studies

The following methodology, proposed by Freivalds (Freivalds, 2009), should be followed when a time study is conducted. The methodology includes all the tasks that should be completed

from throughout the time-study process, from preparation through to the completion of the study.

Firstly, a worker to be studied should be chosen by the supervisor of the particular task. A worker who is expected to have a performance representative of an average performance of his peers is desired, as this will yield the most accurate results. The selected worker should be properly trained in the specific task. It is also important that the operator understands the purpose as well as the methodology of time studies. Furthermore it is beneficial if the operator and the analyst cooperate well with each other. The worker and the analyst need to be able to communicate effectively, and any questions asked by the worker should be answered truthfully by the analyst.

1. The next step involves gathering of information about the task to be studied. The following information should always be recorded by the analyst:
    - The working conditions at the time of the study
    - The tools used including information about the tools as well as the inventory
    - The operators name
    - The analysts name
    - Station where the study is done
2. The third step is to determine where the analyst should be positioned. The analyst should assume a position that is not intrusive and will not interfere with the movements of the operator. Furthermore communication between the analyst and the observer should also be avoided.
3. The task then needs to be divided into subcomponents. These subcomponents should be as small as possible without influencing the accuracy of the study. A guideline is that subcomponents should be longer than 2.5 seconds. The following guidelines should be followed when breaking tasks down into their subcomponents:
    - Both sight and sound should be used to determine the beginning and end of a subcomponent.
    - Manual and machine elements should be kept separate

- Elements that vary in time to complete and those that do not should be separated.
- If a movement is repeated do not record the data. Refer to the first repetition.

4. The study can then be started and one of two timing techniques can be used:
    - Continuous timing - The clock is reset at the beginning of a new subcomponent
    - Snap-back timing - The cumulative time is recorded

2.3. Joint-Tracking and Pose Estimation Using Computer Vision

The Kinect is a gaming peripheral released in late 2010 by Microsoft for the Xbox360 video game platform. The Kinect consists of an RGB camera of VGA resolution and an infra-red depth-sensing camera of VGA resolution with 11-bit depth sensitivity and a microphone array. The purpose of the Kinect is to enable a "controller free gaming environment", according to (PrimeSense, 2011), in which users use their own bodies to control the gaming environment. The Kinect uses structured infra-red light patterns to create a dense 3D pattern and executes a sophisticated parallel computational algorithm to decipher the received light coding and produce a depth image of the scene (Wired.com, 2011). The Kinect is the first consumer-grade application of this technology, according to Kinect hardware developers, PrimeSense (PrimeSense, 2011).

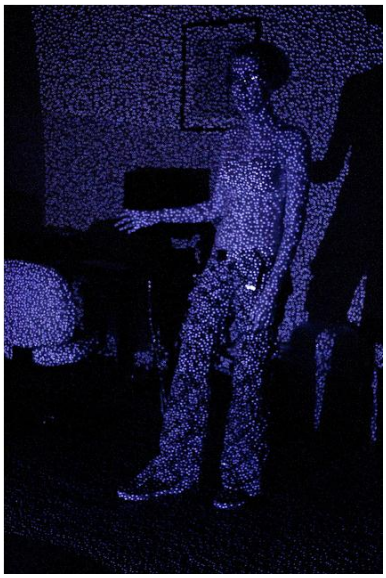


Figure 2: The Kinect's Pattern Projection as Seen By an Infrared Camera ***(Photograph: A. Penven, some rights reserved)***

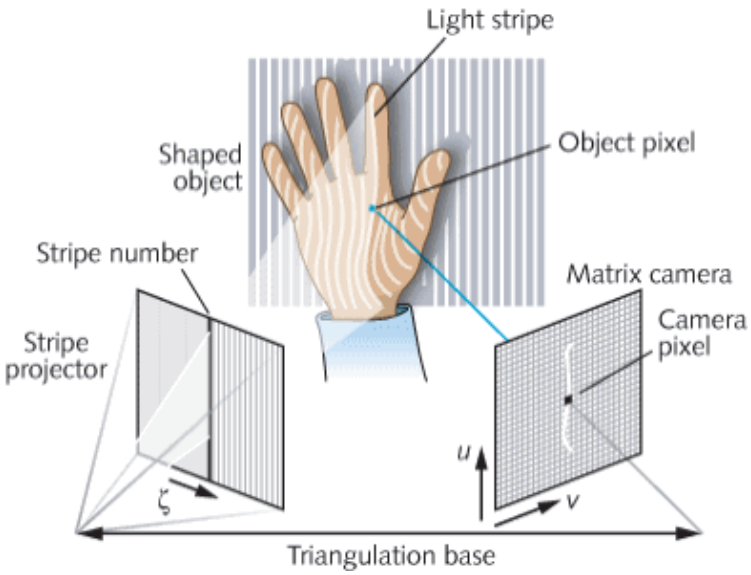


Figure 3: Structured Light 3D Scanning by Measuring Distortion in Projected Stripe Patterns (source unknown)

Microsoft created the Kinect camera to enable users to control the Xbox entertainment system using their bodies, by tracking joints on the players' bodies, and inserting avatars of them into games, or using hand position as a mouse-pointer. The ease of tracking players is increased due to the fact that players are easily removed from the rest of the scene, and that 3D positional points add an extra dimension of data which can be used to accurately make a pose classification.

The joint-tracking used by the Xbox is a sophisticated per-frame, per-pixel classification system utilizing decision forests and a training set of over 500 000 images. Microsoft Research Cambridge, the creators of this joint tracking system, state that consumer-level joint-tracking was made possible by the introduction of inexpensive 3D cameras such as the Kinect. Microsoft Research has since released parts of their code to the public, for 3rd-party application development and for use by the scientific community (Microsoft Research Cambridge & Xbox Incubation, 2011).

3. Real-Time Performance Feedback and Time Study Using Computer Vision Technology

3.1. Consideration of Alternative Methods

3.1.1. Computer vision: Hardware

This chapter will explain the considerations in choosing the hardware used in the system. The vision systems (cameras) considered were:

- Webcam
- Microsoft Kinect Camera

3.1.1.1. Webcam

Previous work by Van Blommestein (Van Blommestein, 2010) shows that a webcam can be used to perform hand tracking. Sophisticated computer-vision algorithms exist which can determine the locations of some joints. The simplest of such recognitions is performed by placing colour-coded markers on the subject's body, and applying a colour filter, which then filters out the rest of the scene and outputs only the location of the marker.

Figure 4: An actor wears a motion capture suit which enables skeletal tracking for character animation (source: Internet)

Some advantages of using webcams are:

- Cost: Webcams are very affordable

- Connectivity: Nearly modern PCs have support for webcams and many may include the necessary drivers which eases installation.

There are however, disadvantages to using webcams:

- Lighting: The scene to be analysed requires adequate lighting. Shadows, lack of contrast or brightness, and colour interference may hinder the system's ability to perform recognition
- Reliability: Markerless recognition may be unreliable, depending on the algorithms employed
- Computational Cost: There is typically a lot of information in a scene which needs to be analysed. If this analysis is performed entirely by software then there is a high computational cost involved.
- Intrusive nature of markers: Van Blommestein (Van Blommestein, 2010) showed that tracking could be performed in an industrial environment by having workers wear coloured gloves. Workers found the gloves to be intrusive, adding discomfort and impeding work performance slightly.

3.1.1.2. Microsoft Kinect

The Kinect camera is a recent development in human joint tracking, and the first solution which introduces this technology to consumer applications. The Kinect camera captures a depth scene using infrared light. This depth image allows for easy differentiation of a human subject from the scene, and an extra data dimension to increase the success of pose recognition. The Kinect does not rely on natural lighting to render a scene, using instead on its own infrared light source, and can thus be used in all lighting situations (PrimeSense, 2011).

For this project, the Kinect was chosen as the primary vision system. The main reason for this choice was the novelty of the technology, and the potential for marker less tracking.

Some of the disadvantages of using the Kinect are that it is more expensive to purchase than a webcam, (though it is in the opinion of the author still affordable). The price of the Kinect is offset further by noting the fact that it can potentially track up to 15 people in a scene. Though

it would be unpractical to track that many due to space requirements within the viewing frame, simultaneously tracking up to three subjects using one camera is certainly feasible. It should be noted that the system developed for this project tracks only one user. Another disadvantage is that currently the system requires an initial calibration pose by the subject in order to begin the joint-tracking. The calibration process takes approximately 3 to 10 seconds, and requires that the subject assumes a certain pose. This may be seen as intrusive or stress-inducing by workers, though the intrusion is negligible in comparison with previous solutions which required that workers wear special clothing. In many situations calibration can be bypassed, for example when a worker returns to a task calibration does not have to be performed again.

3.1.2. Computer Vision: Software

Two options currently exist for software which can perform human joint tracking using the Xbox Kinect camera:

3.1.2.1. Microsoft Research Kinect for Windows SDK

Microsoft released a non-commercial Kinect software development kit for Windows on June 16, 2011, in order for academics and enthusiasts to build applications using the Kinect. A commercial version is planned for a later date and has not yet been released at time of writing. Kinect for Windows SDK allows access to:

- Raw data streams: Access to low-level streams from the depth sensor, color camera sensor, and four-element microphone array.
- Skeletal tracking: The capability to track the skeleton image of one or two people moving within the Kinect's field of view.
- Advanced audio capabilities: Audio processing capabilities include sophisticated acoustic noise suppression and echo cancellation, beam formation to identify the current sound source, and integration with the Windows speech recognition API (Microsoft Research, 2011).

3.1.2.2. OpenNI and NITE Software

The OpenNI organization is an industry-led, not-for-profit organization formed to certify and promote the compatibility and interoperability of Natural Interaction (NI) devices, applications and middleware (OpenNI Organization, 2011) In December 2010, the company PrimeSense, whose depth sensing reference design Kinect is based on, released their own open source drivers along with motion tracking middleware called *NITE*.

Like the Microsoft Kinect for Windows SDK, this SDK allows for access to low level streams from the depth sensor and colour camera sensor, but features a slightly less sophisticated skeletal tracking system which tracks fewer joints and requires an initial calibration pose by a user. The NITE skeletal tracker, however, allows for torso-only tracking, something which the Microsoft Kinect for Windows SDK has yet to implement.

3.1.3. Cycle Recognition

The nature of this project is that cyclic, repetitive work is performed by a worker, and that the cyles need to be classified as productive work. Previous work by Van Blommestein (Van Blommestein, 2010) showed that productivity could be estimated by observing a workers level of activity over an extended period. This estimation is improved by the system presented in this project, which assumes that a work part has been completed or assembled when a specific cycle or pattern is recognized in the workers movements.

The improvement which cycle recognition offers over simple activity classification is clear: Actions are identified with greater accuracy and there is a far lower likelihood of incorrectly classifying an action as a productive one (type II error).

Machine learning is also implied in this project, as the system should be able to easily learn a new work sequence with little or no input from the operator.

Finding a suitable cycle recognition method was one of the biggest challenges for this project. The greatest difficulty lay in the novelty of the problem – there was little existing work which could be compared to the problem, and work which did exist proved unhelpful, as it was either limited in some way or was too complicated for the scope of a final-year project. Thus a new

solution had to be found which could be implemented. Some of the criteria which were considered for cycle recognition algorithms were:

- They had to be trainable and be able to perform recognition
- They had to be computationally simple, so that they could run in real-time at up to 30 frames per second
- They had to be time-invariant, that is, the speed at which an action is performed should not affect the recognition
- They should be able to cater for short actions of a few seconds, as well as long actions of several minutes in length without becoming computationally inefficient over time
- They should be able to learn and discern between a number of different actions
- They should be simple to understand, such that their implementation is within the scope of a final-year project

The general methodology followed is given in Figure 5 below:

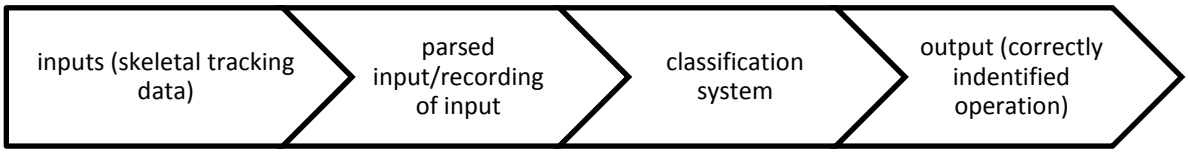


Figure 5: Cycle Recognition Methodology

The approaches considered during the course of research for this project were:

3.1.3.1. Hidden Markov Models

Hidden Markov models (HMM) work well with time series data, which is why they are typically used in motion analysis (Dreijer J. , 2011). Hidden Markov models are easy to conceptualize, which eases their design, but training the HMM may be more difficult than an artificial neural network. HMMs could not be used to the difficulty in implementation. Future work may be based on HMMs, which could lend improvements to some parts of the recognition and learning process, by for example, allowing for skipped node visits, more flexibility in training and the introduction of probabilistic detection.

3.1.3.2. Artificial Neural Networks

Artificial neural networks (ANN) are inspired by the biological neural structures found in the human brain (Nissen, 2009). They consist of an interconnected group of nodes, akin to the vast network of neurons in the human brain. They are usually used to model complex relationships between inputs and outputs or to find patterns in data. Artificial neural networks are easy to train and are typically, flexible and robust, though they are difficult to set up according to (Nissen, 2009). A drawback to using artificial neural networks is that they are unforgiving if not set up correctly, and in the case of this project may be very difficult to troubleshoot. In most cases, ANNs are given second preference to HMMs in cases that deal with time-series data, as ANNs are most suited to dealing with massively parallel calculations. ANNs have been used by (Wassner, 2011) to perform gesture recognition and learning, which is why they were considered for this project.

3.1.3.3. Frequency Domain Analysis

By realising that work is repetitive in nature, and the same actions will be performed periodically, the discrete Fourier transform (DFT) of relevant joints' positions over a few cycles could be used. This will transform the joints position with regards to time into a function of frequency. One of the biggest challenges for recognition is being able to recognise both very fast and very slow actions accurately, since time-shifted actions may look very different using any other method. Using DFTs proves very useful, as a discrete, frequency representation of movement is time-independent. Furthermore, the DFT can be used simultaneously to determine the frequency of the work being performed (the workers performance), as well as recognise the task being done. A drawback to using this method is that multiple cycles will need to be realised before prediction of tasks can be made, so that breaks between work-periods may not be accurately described. To the author's knowledge this method has not been used before for gesture recognition.

3.1.3.4. Finite State Machine

A finite state machine (FSM) is a behavioural model used to design computer programs. It is composed of a finite number of states associated to transitions. A transition is a set of actions

that starts from one state and ends in another (or the same) state. A transition is started by a trigger, and a trigger can be an event or a condition. In the case of this project, "states" would be pre-defined zones in the worker's workspace, and the worker would transition through these states as he passes his hands through them while he works. When the FSM reaches its final state, the time is recorded and in this manner the time taken to perform an action can be determined.

Finite state machines are the simplest method of those described here, both to visualise and to implement, however they are limited in a number of ways:

- They require a separate process to determine where the "states" or zones are. These can either be input manually by the operator, by visually demarcating the workspace, or by some learning algorithm.
- There is no tolerance for skipped states. If the system fails to register that one state in a sequence was skipped, then the entire sequence will have to wait until that state is reached in the next cycle, potentially allowing for inaccurate times to be logged.
- Training an FSM to learn a specific sequence is usually done prior to the compilation of a program or is input manually by an operator. In this case we would like the FSM to learn new sequences during the course of the program as there should be as little operator involvement as possible throughout the entire operation. Furthermore, the number of states in a finite state machine is inherently implied as being "finite", but we would like the system to be able to learn a sequence consisting of any number of states, which requires some other solution.

3.2. Choosing Alternatives and Planning Methodology

As mentioned previously, the primary goals of this system are:

- To provide constant performance feedback to a worker performing a repetitive task
- To partially automate the time-study process

In previous sections, these goals were translated into the following methodology:

- Use computer vision technology to track a worker's action, analyze this action and classify it as a productive action
- Provide feedback to the worker which is easy to understand and does not distract from work
- Log performance in such a manner that it can be analysed at a later stage by a time-study operator

This section will explain the choices made in selecting technology and methodology for this project, in order to fulfil the requirements set above.

3.2.1. Hardware: Microsoft Xbox Kinect Camera

The Xbox Kinect camera was the fundamental technological innovation which led to this project. The ability to perform marker-less skeletal tracking is seen as a big incentive to use this camera, and which offsets the increased cost, when compared to standard webcams. Natural interaction is a rapidly developing field with applications in many fields, and this rapid advancement has a great deal to do with the excellent hardware and software which is now available.

i) Connectivity:
   - The Kinect has a standard USB connection, enabling connection to and installation on any modern PC. (PrimeSense, 2011)

ii) Joint tracking:
   - Middleware offered by the OpenNI group provides joint-tracking functionality to the Kinect, enabling developers to track 20 joints on the human body in three dimensions, with calibration for absolute coordinates. (Wired.com, 2011)
   - Joint tracking greatly simplifies gesture recognition and enables greater accuracy and reliability in recognition. (Wired.com, 2011)
   - The tracking is completely non-intrusive. Currently an initial calibration pose is required, though future refinements may eliminate this need. No special clothing is required by the operator.

iii) Nature of Technology

- Due to the high sales volume of the Kinect and the ease of installing the open-source drivers and API from OpenNI, the Kinect has become a popular choice for many person-tracking operations. This implies a large support-base and the availability of open source software.

iv) Price and Availability

- The Kinect is currently the only commercial, non-intrusive skeleton tracking system available, according to (PrimeSense, 2011). The Kinect is available from many retailers in South Africa due to its affiliation with the Xbox gaming console.

The developers of the Kinect, PrimeSense, have teamed up with manufacturer Asus in order to produce the *Wavi Xtion*, which is in essence a PC-compatible device similar to the Kinect, and using most of the same hardware and PC drivers, although there is no compelling reason to choose it over the Kinect camera, besides the fact that it is currently intended solely for application development where the Kinect camera is still just a gaming peripheral. It is currently available from Asus South Africa. (Asus, 2011)

This development will eventually ensure that the Kinect's technology will find its way into other uses, besides being a video game controller, and should mean a reduction in cost as more 3rd-party manufacturers start to produce products with the same technology.

3.2.2. Software: OpenNI SDK with NITE Middleware

Although the Microsoft Kinect for Windows SDK has a more sophisticated method for performing joint tracking which does not require calibration, tracks more joints, and is generally more accurate, it currently does not facilitate torso-only tracking, instead expecting the entire body of a subject to be visible within a scene. As it is foreseen that most work operations will be performed by workers who are either sitting or standing behind a workbench and have their lower bodies obscured by a work surface, this is an impediment which cannot be overcome. The NITE software facilitates upper-body only or hand-only tracking, which is why it was selected for this project.

In testing both systems, it was noticed that the NITE software also outperformed the Microsoft Kinect for Windows SDK in the following areas:

- Performance: Testing both systems on the same hardware showed that the OpenNI SDK had a consistently higher frame-rate when compared to Microsoft's SDK
- Openness of Software: The OpenNI SDK framework is more open than Microsoft's SDK and allows for greater access to most functions
- Documentation and Ease of Programming: Due to OpenNI's earlier release, there was significantly more online support and example code available at the start of this project

3.2.3. Cycle Recognition Algorithm

For this project, it was decided that cycle recognition would be implemented rather than area based recognition, in order to build upon the work of (Van Blommestein, 2010).

In section 3.1.4 the requirements for a method to perform cycle recognition were stated. Ultimately, the Finite State Machine (FSM) was chosen as the best method to use.

In order to describe the choices made it is necessary to break the cycle recognition process up into discrete parts:

3.2.3.1. Parsing Inputs

Inputs from skeletal tracking data consist of a stream of joint coordinates in 3 dimensions (X, Y, and Z). These data need to be parsed by some algorithm before they are useful for recognition purposes. This data was already trimmed before parsing occurred: Firstly, only the joint coordinates corresponding to the user's left and right hands were used, and secondly, only two dimensions were considered (X, Y) in order to simplify calculations. A third dimension was considered superfluous. Methods considered for parsing inputs were:

- Time series: This method simply adds a time stamp to the two dimensions (X, Y, t) and stores data in an array.
- Frequency domain analysis: This method attempts to extract frequency patterns from movements by observing that all work performed is cyclical in nature. A fast Fourier transform (FFT) could be used to uncover these patterns, but pre-parsing would have to occur in order to determine what the inputs to the FFT should be in the first place. No solution to this fundamental problem was found.

- Node Generation: This method is suited to an FSM, as described in section 3.1.4.4. The system learns where significant zones in the workspace are and logs visits to that zone. For example, if a worker repeatedly reaches for a bin, the location of that bin is stored and an event created every time the system recognizes that the worker has reached for that bin.

A combination of frequency domain analysis and node-visits was also considered; the relative frequencies of node visits could be assessed and a pattern found from this data.

The chosen method to parse inputs is node generation, as this was the method most suited to the FSM approach to cycle recognition.

3.2.3.2. Learning Algorithm

A requirement for this project is that operator input should be minimal, and that the system should be easy and transparent to set up and configure. This requirement was translated into a need for the system to be able to learn different actions on the fly, without any prior information being known about these actions and without having to configure the system for any specific actions.

The finite state machine (FSM) approach which was taken necessitated a simplification of this requirement, as it is implied in a "finite" state machine, that the number of states in a sequence is finite, and therefore difficult for the system to learn on the fly. Thus it was found necessary for the number of states to be input by an operator prior to the running of the program. The "number of states" in this case translates to the number of events recorded by the program, where an event is defined as the operator placing his hand into the area defined as a node.

The approach taken was to record the sequence of events once, after a training period to ensure that the system was settled. This approach requires some care by the operator, as if a sequence is learned incorrectly it will not be corrected and the learning process would have to be reset manually.

Other approaches considered included modelling the cycle as a Markov process, with probabilities of state-transitions settling after a number of repeated cycles. This approach may

be favourable to the one used in this project and should be considered as a future improvement, though there are complications involved.

3.2.3.3. Recognition

Due to the FSM approach taken, recognition of cycles was relatively simple. The method chosen was a "subsequence" method. A subsequence of a sequence is a subset of the elements in a sequence, where the order of the elements is preserved. In this case, the *sequence* is a recorded array of all events, and the *subsequence* contains the sequence of events which need to be recognised.

Other methods considered were probabilistic in their recognition, which would potentially allow for skipped node visits. The method proposed in this project assumes that nodes will not be skipped, though additional nodes may be visited in error by the worker without disrupting the cycle.

3.2.4. Summary of Cycle Recognition

In summary, the following approach was chosen:

1. Create an algorithm which calculates where important nodes are in the viewing field
2. Assign values to these nodes (A, B, C ... n)
3. Create an event each time a node is visited and log these visits
4. Extract the cyclical pattern from the logged node visits and teach this pattern to a finite state machine (FSM)
5. Observe subsequent events which transition through the FSM and log whenever the FSM has reached its initial state (i.e. when a cycle has been completed)
6. Compare times logged between subsequent cycles in order to obtain cycle time

These steps are explained in greater detail in the sections following.

3.2.5. Performance Calculation and Feedback Method

Performance calculation can be performed in two different ways:

When the standard time is known, a simple performance calculation can be made which expresses the observed time as a percentage of the standard time.

Where standard time is not given, observed time can be compared to an average of previously logged observations recorded during the running of the program, and calculated as a percentage of the average of previously logged observations.

This system uses a 15 inch colour LCD display to display feedback to the worker. Feedback should be clear and it should not require more than a glance for the worker to interpret it, as that would cause the feedback to distract attention from the task being performed. A simple colour scale will thus be used: for performance calculation of ≥ 100%, the screen will be green, indicating favourable performance, and for diminishing performance the green will transition through yellow into red, becoming purely red at some calculated or estimated lower threshold.

3.3. Execution Methodology

An outline of the intended computer system foreseen for this project is shown in Figure 6:

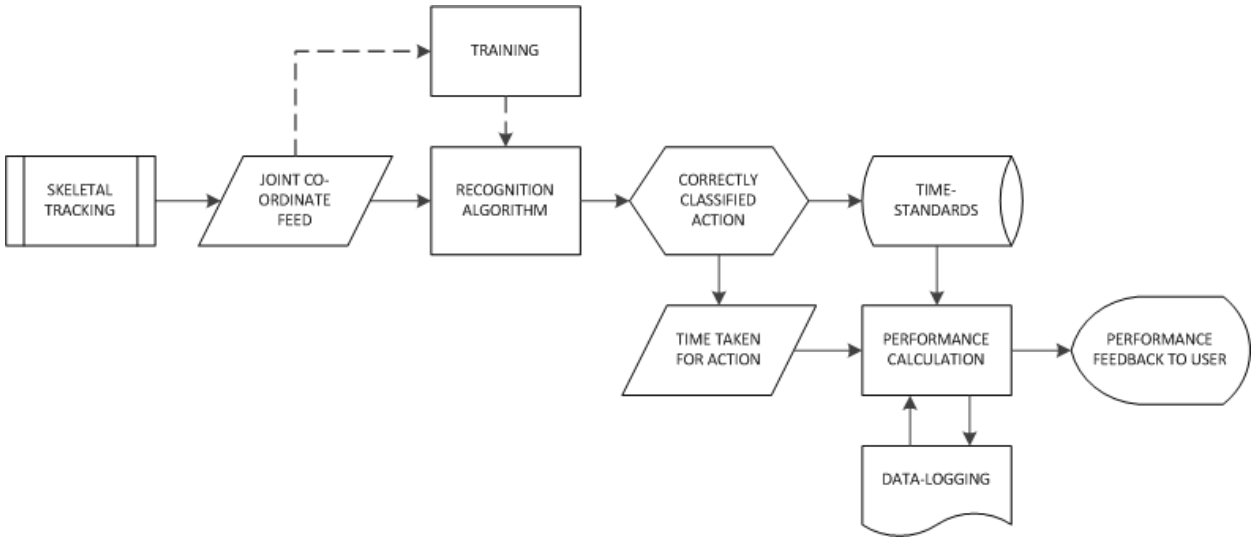


Figure 6: Steps in Execution of Cycle Recognition Project

Each aspect of this system will now be discussed in detail

3.3.1. Computer Vision System

The Kinect for Microsoft Xbox 360 is used in this project. This section will explain the set up and implementation of the Kinect camera.

Figure 7: The Microsoft Xbox Kinect Camera (iFixit.com)

3.3.1.1. Power and Connection to PC

The Kinect camera, shown in Figure 7, connects to a PC via a standard USB port, but requires an additional external power source. A standard SABS 164 plug is supplied for Kinects sold in South Africa, which is connected to the AC mains.

3.3.1.2. Imaging Hardware

The Kinects hardware system layout is shown in Figure 8 below. To be noted is that a lot of the image processing happens on board the Kinect by the PrimeSense PS1080 microchip.

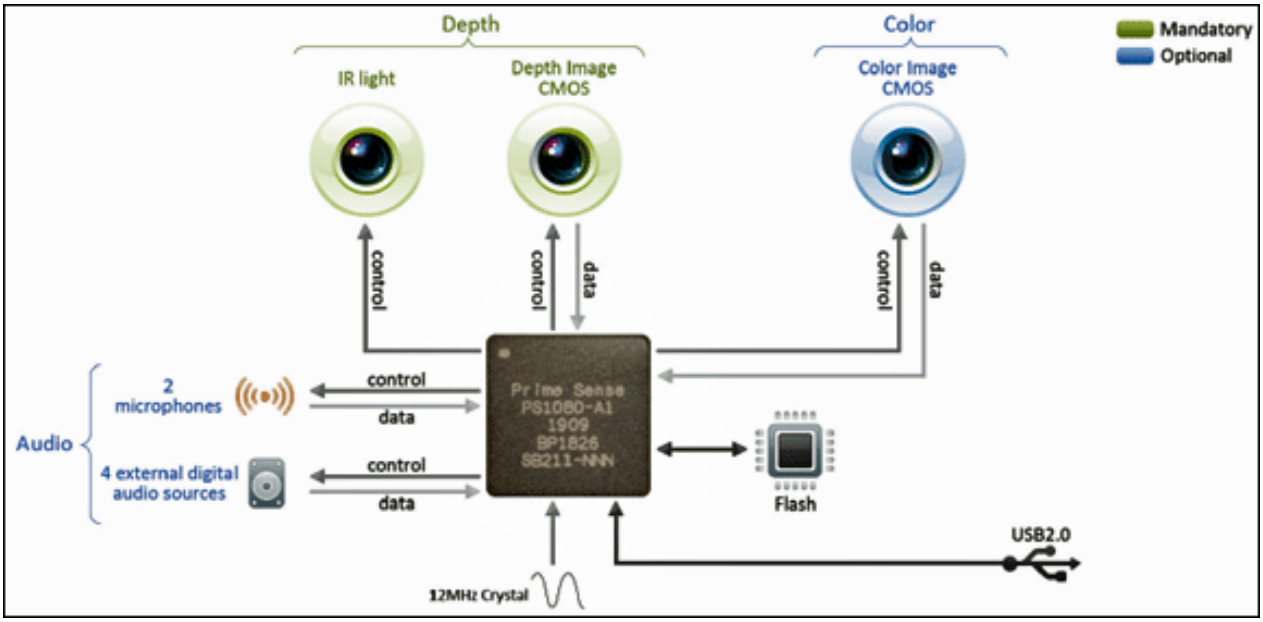


Figure 8: Internal Processing Layout of the Kinect (Source: PrimeSense)

The Kinect interprets 3D scene information from a continuously-projected infrared structured light. The infrared light projects a structured pattern onto a scene, invisible to the naked eye, and the depth image CMOS sensor. The Kinect encodes information in light patterns as it goes out, observes the deformations of that light pattern, and then uses sophisticated processing on board the Kinect to produce a three-dimensional scene from the data.

3.3.1.3. Software Installation

Software installation is detailed in the appendix.

3.3.1.4. Physical Installation

The Kinect is placed on a tripod, facing the worker, at a distance of 2 to 4 metres. It should face downwards at an angle of approximately 20 to 30 degrees. The worker's entire upper body, including head, should be included in the scene.

3.3.1.5. Scene Information

a. Depth Scene

Figure 9: 3D Scene as Viewed by the Kinect (image source unknown)

OpenNI's Scene Analyzer analyzes a scene, including the separation of the foreground from the background, identification of figures in the scene, and detection of the floor plane. The Scene Analyzer's main output is a labeled depth map, in which each pixel holds a label that states whether it represents a figure, or it is part of the background.

b. User recognition

As seen in Figure 10 below, each figure identified in the scene is coloured in a different colour, and the User Generator searches for the calibration pose. Once the figure is in the calibration pose, calibration is performed on that figure. When the calibration is successfully completed, a skeletal representation of the figure exists.

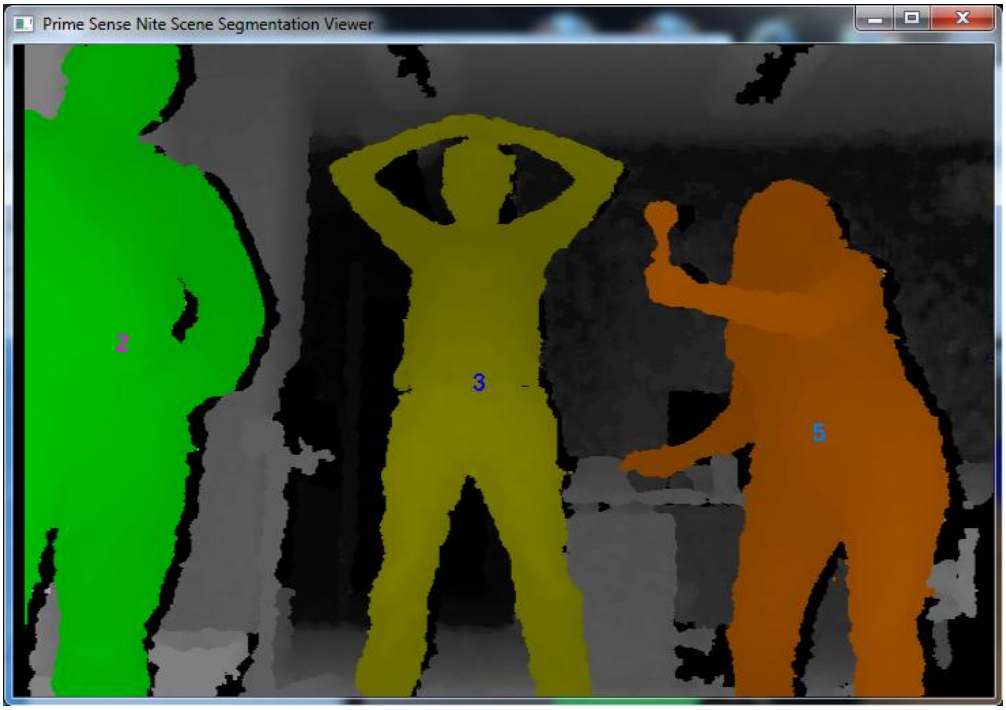


Figure 10: Screen Capture of OpenNI Scene Analyser Performing User Identification

3.3.2. Skeletal Tracking

In order for the *NITE* software to perform Skeletal tracking on a user, the user must first perform assume a calibration pose. The calibration process captures and analyses the various proportions and measurements of the figure's body, to optimize the specific tracking of its movements. Calibration information can also be saved and loaded later however, removing this requirement. Workers of similar height and proportions would, for instance, not be required to perform calibration, as would a worker returning to his workbench.

3.3.2.1. Joints Tracked

The Skeletal Tracker recognizes 14 joints, namely: Hands, elbows, shoulders, hips, knees, feet, head, neck, and torso. Figure 11 below shows a user performing the required calibration pose, with the skeleton joint data superimposed onto the user's body.

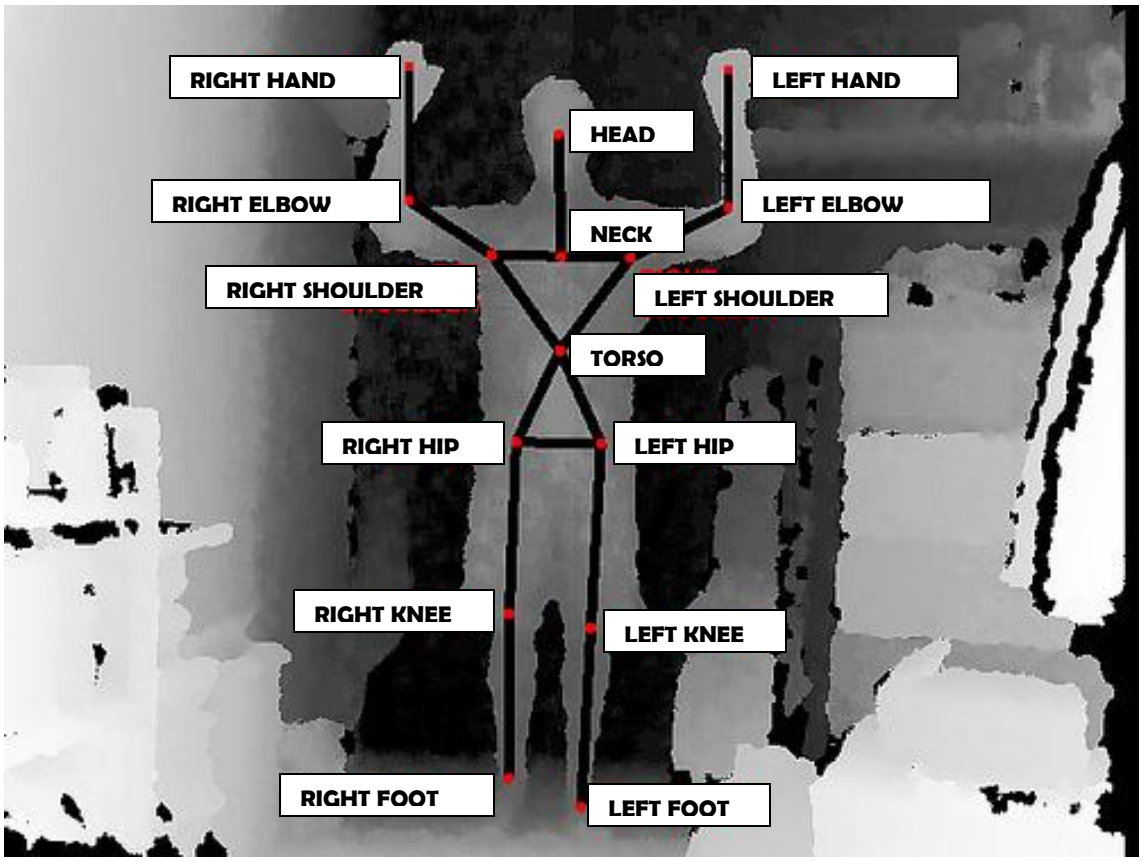


Figure 11: Skeletal Tracking Showing Joints Tracked (source: PrimeSense)

The system used for this project uses coordinate data from only the left and right hands, though it is necessary to track the skeleton throughout the operation to obtain these coordinates.

3.3.3. Joint Coordinate Feed

The basic data for the system are obtained from a joint coordinate feed, supplied by the skeletal tracker. The data from the feeds is provided in real world coordinates, in millimetres, which are then projected into screen coordinates, given in pixels. This is done using a built OpenNI function. This system uses only the X-, and Y- coordinates of the user's hands, as it was reasoned that tracking third-dimension was unnecessarily complicated and that it did not add any value. Thus the Z- coordinates are discarded.

3.3.4. Training

The training process used in the system follows the following process:

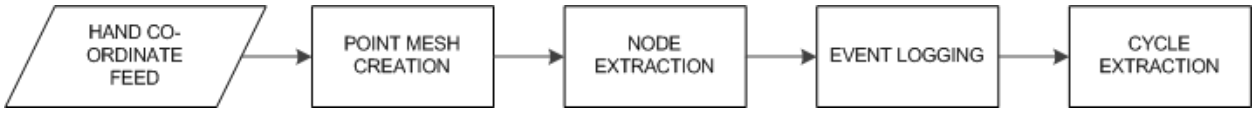


Figure 12: Methodology of Training Process

3.3.4.1. Heat map

First, a two-dimensional array is created, which will hereafter be referred to as the "heat map". The data which is collected from the joint coordinate feed is projected into a screen resolution of 640x480 pixels, and as this resolution is considered unnecessarily high, the data collected by our heat map is reduced by an arbitrary factor to reduce this resolution. The system reduced the resolution by a factor of 8 for the heat map, resulting in a field of 80x60 points.

The aim of these points is to collect data from the scene about where the user's hands spend the most time, as these points are the points the user visits in each cycle. For example, a user performing an assembly task removes items from 3 bins located on the workbench in front of the user. The aim of the heat map is to extract the locations of the bins by finding locations where the hands spend the most time. These pseudo bin locations are hereafter referred to as "nodes".

A spread sheet mock-up of an example heat map is shown in Figure 13, with the significant points circled in red. The value of each cell represents an accumulative score, which is a function of the time the user's hand has spent in that cell.

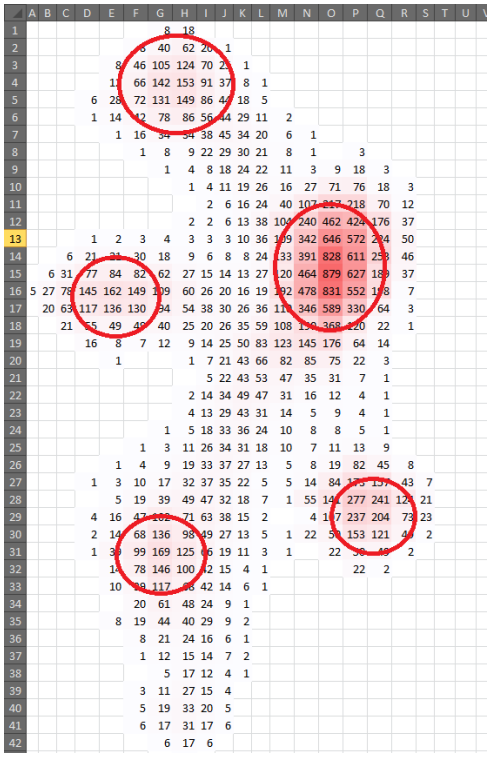


Figure 13: Mock Up of Field Mesh Using Spread Sheet Software

Each frame, the coordinate data is received and parsed, and the cell represented by the coordinates of the users hand, and its surrounding cells, are given a score calculated in the following manner:

$$\mathrm{N} = \frac{\text{time in milliseconds since last frame}}{50}$$

The factor 50 in the previous equation is arbitrary as nothing in the heat map is absolute. The cell and surrounding cells are then given a score by multiplying N as shown in Figure 14, where the cell at the centre is the cell in the heat map representing the user's hand's current coordinates.

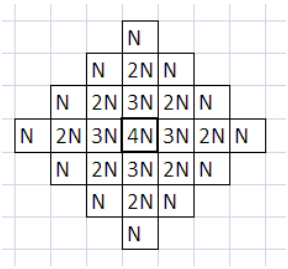


Figure 14: Distribution of Values for Calculating Field Mesh

3.3.4.2. Determining Nodes

The system locates the highest values in the heat map in order to determine where the significant nodes are. This problem is reduced to an optimization problem where multiple local maxima are sought. The most reliable method found was a simple highest neighbour search, where each point is compared to its surrounding points and if none of its surrounding points have a greater value than itself, then that point is a local maximum.

The drawback of using this method is that it is a brute-strength approach and is very expensive computationally. For example, if comparing each cell in our 80x60 heat map to the cells ≤ 3 cells away, the computational cost is calculated as:

$$80 \times 60 \times 7 \times 7 \; = \; 235\,200 \; calculations \; per \; frame$$

When considering that the program runs at 20 frames per second or more, this cost was far too high and reduced the program's frame rate. To reduce the cost the highest neighbour search is run every other frame instead of every frame, and only values which are large relative to the absolute maximum of the mesh are checked.

In this way, this brute-strength algorithm is able to run in real-time without hindering the performance of the system.

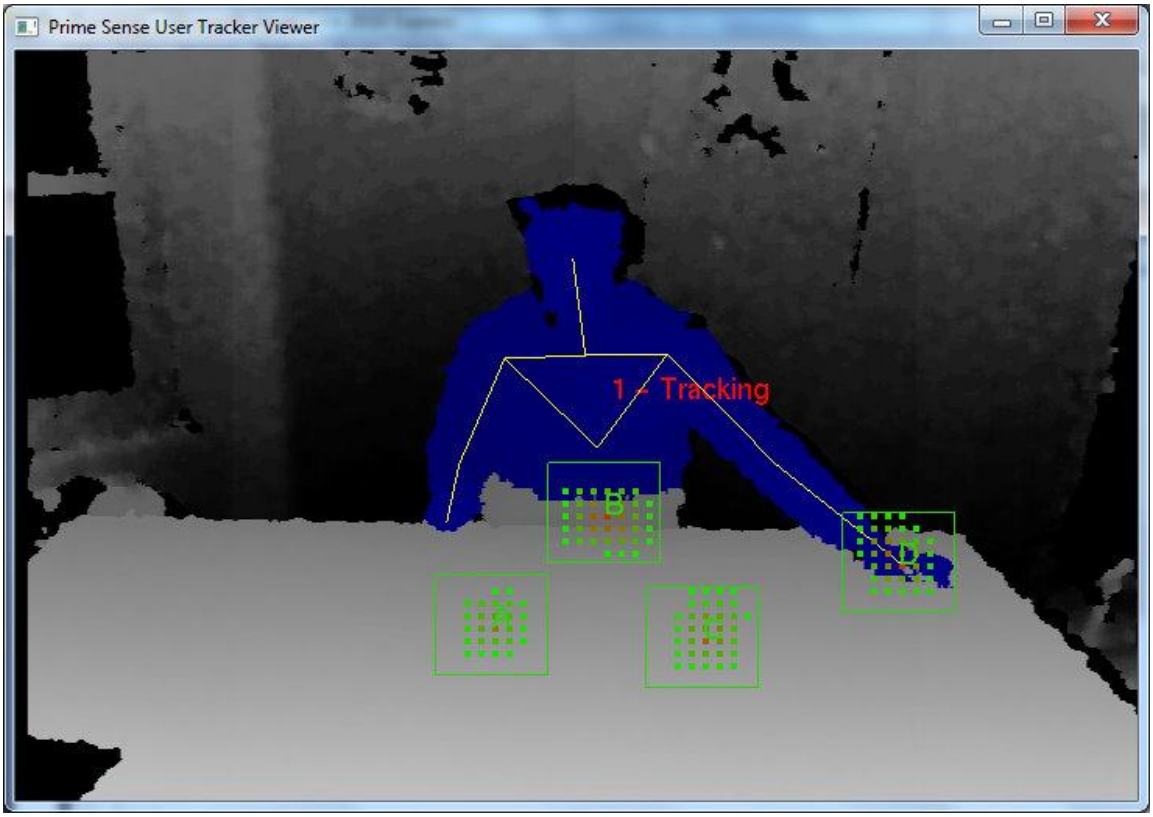


Figure 15: Screen Capture of Node Identification Process

The next step is to determine whether the system has achieved a "steady-state". That is, the system needs to perform a check in order to ascertain whether the nodes it has found are indeed the correct nodes, indicating that the node-finding part of the training algorithm has been completed.

The safest method to do this is to input the number of nodes into the system before execution, and then assume that the system will undergo a learning process where at first there will be very few nodes, then the number of nodes will rapidly increase, and then after a number of repetitions of the work-cycle slowly reduce to the correct number of nodes. Once the correct number of nodes has been found and some time has passed, the system is assumed to be in steady state. This method was not used however, due to the requirement that the system should require as little operator involvement as possible.

To offset the requirement that the operator inputs the number of nodes beforehand, a different method was devised, which uses the cycle recognition algorithm as described in the

following section (3.3.4.3). During the training time the worker should perform only the task which the system is trying to learn, and should not pause or place his hands in positions other than the nodes. Thus for this method an operator should keep watch during the training process to ensure that the system is trained properly.

Once the system is assumed to be in steady state by the program, the nodes are written to an array and are fixed to their positions. The nodes found are stored in an array and labelled A, B, C ... n.

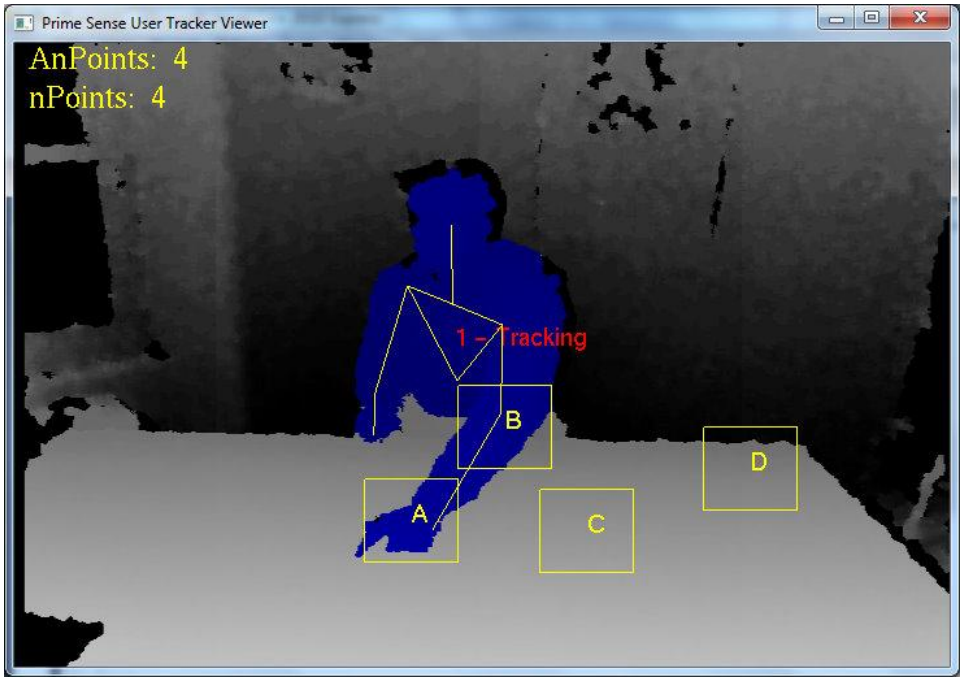


Figure 16: Screen Capture: Training Complete and Bin Locations Fixed

3.3.4.3. Learning a Cycle

Once the nodes are found and are fixed, the system needs to learn the work cycle which the worker is performing so that it may perform cycle recognition.

The system starts this recognition by recording node visits as a string of characters, with the nodes being labelled A, B, C etc. This string is then checked for patterns by a pattern detection algorithm. The problem is thus reduced to the following description:

"Find the longest continuously repeated string within a string, which doesn't consist entirely of a repetition of a string"

As an example, we take the input string **ABABC ABABC ABABC ABABC**. First of all, we notice **AB** is repeated, but **AB** is not the solution because there are other substrings **ABABC** and **ABABC ABABC** which are longer and are also repeated. We also can't have **ABABC ABABC** because that consists entirely of another repeated substring, i.e. 2x **ABABC**, leaving only **ABABC** as a solution.

There are a few different approaches one can take to solve this when dealing with a clean input string which consists only of the pattern, however in this system it must be foreseen that this will not be the case, and thus other provisions need to be made.

The solution implemented uses regular expressions to extract the cycle from a string. The following regular expression was derived for this project:

```
(.{3,}?)\1{2,}
```

The parts of this expression are:

- `.{3,}` – match any group of 3 or more characters
- `?` – find the shortest match
- `\1{2,}` – match the preceding expression two more times

The expression thus translates entirely to: *"Match the shortest group of 3 or more characters which is repeated more than twice."* which is a neat description of what the recognition algorithm is required to do. It is assumed here that a cycle will always consist of 3 or more actions.

3.3.5. Recognition

The method used for recognition consisted of a simple substring search.

A substring is defined as a "subset of the symbols in a string, where the order of the elements is preserved". For example, the string "**N**-**I**-**C**-**E**" is a substring of "A-**N**-**I**-**C**-B-D-A-**E**-G".

All node visits are recorded into an "events" string. The cycle string learned during the training phase of the program, or "action" string is then searched for within the events string. When the action string is found within the events string, it means that a complete cycle has been performed.

This method simplifies the concept of a finite state machine, as it is very easy to specify new action strings, which eases the task of programming. It has the additional benefit of ignoring incorrect node visits – as long as all the nodes have been visited in the cycle, a cycle is recognized.

Substring search can also be used to identify any number of different action strings, so the program can recognize a number of different actions without needing to be reconfigured.

3.3.6. Performance Calculation

Performance calculation is performed in one of two ways:

1. By comparing the cycle time to a standard time, which is input by the operator before the execution of the problem.
2. By comparing the cycle time to the average (*A*) of all previous cycle times. The average of *n* cycle times (*CT*) is calculated by:

$$A_n = \sum_{i=1}^{n} \frac{CT_i}{n}$$

To avoid having to store too many values during programming this is simplified to:

$$A_n = \frac{n(A_{n-1}) + CT_n}{n}$$

For both methods 1 & 2, the performance P_i is then calculated as a percentage by:

$$P_i = \frac{CT_i}{ST} \times 100\%$$

Where *ST* is either the calculated average time (*A*), or the standard time which is input by the operator.

3.3.7. Feedback to Worker

Performance feedback to the worker is kept as simple as possible in order to avoid distraction. The basic principle was a Red light, Yellow light, Green light method, where green indicates a favourable performance, and red indicates inadequate performance. This is a simple and intuitive system which is easy to explain to a worker, and does not require more than a quick glance at the screen by the worker in order to receive meaningful feedback. A screen capture of the performance feedback screen is shown in Figure 17 below.

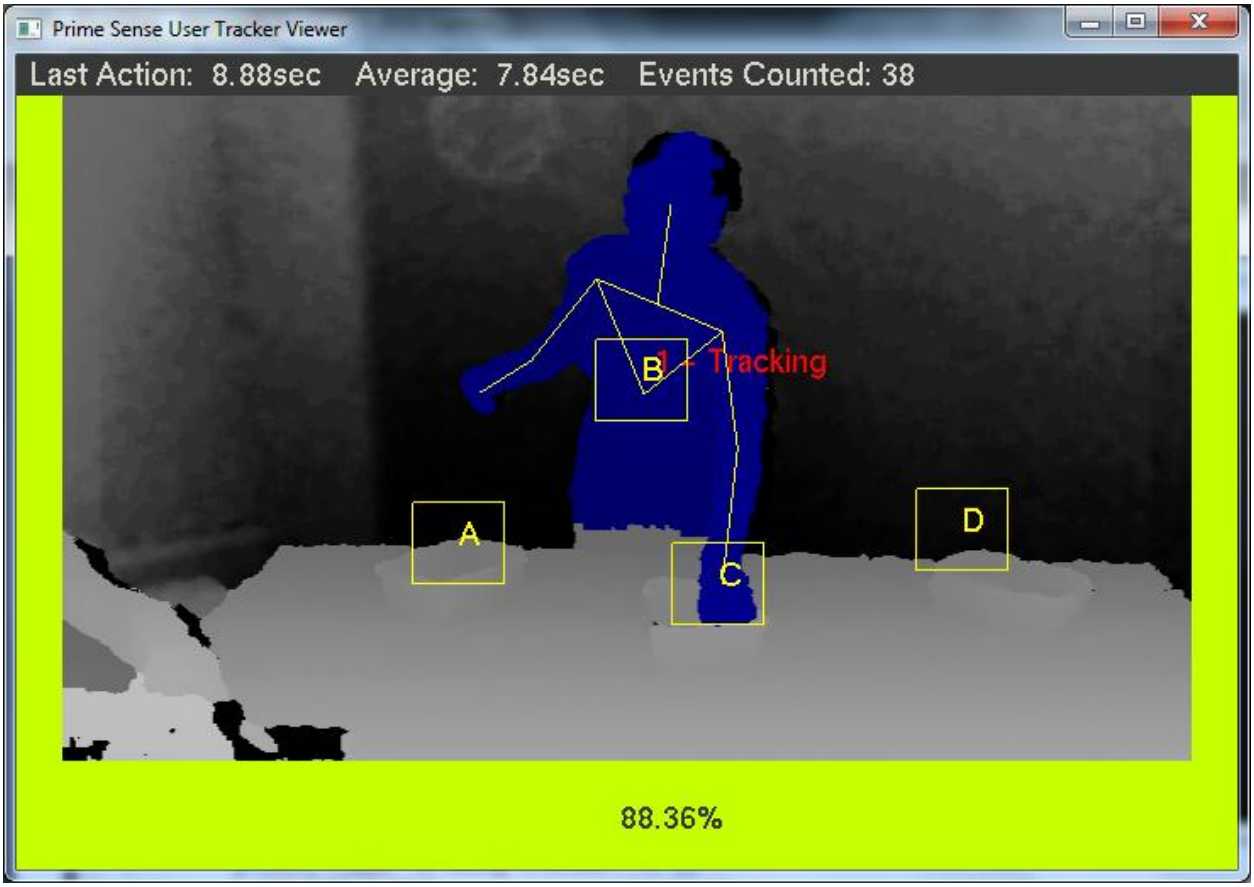


Figure 17: Screen Capture of Output of a Peformance Feedback Screen (Image by Author)

The feedback colours were calculated by the following scale:

70	72.5	75	77.5	80	82.5	85	87.5	90	92.5	95	97.5	100

Figure 18: Colour Scale Used for Performance Feedback

3.3.8. Data Logging

Data logging is performed by a simple `fprintf` statement in C++, which provides the analyst with a tab-delimited text file (`.txt`) with the following columns:

- **Action recognised** – In order to discern between different cycles, should the worker start a different task during operation
- **Cycle duration**– Calculated as the difference between the previously logged time and current time
- **System Time** – To provide an absolute time reference for the analyst

Analysis of this data can then be performed using Microsoft Excel or MATLAB.

4. Demonstration of Method

This purpose of this chapter is to assess the effectiveness of the system proposed in this document. In order to determine the effectiveness of the proposed solution, a simple assembly task was designed as an experiment. Only the time-study aspect of the project was assessed, as the effectiveness of performance feedback is difficult to quantify in an experiment.

4.1. Aim

To determine the effectiveness of the automated time-study system proposed in this project.

4.2. Method

The effectiveness of the system will be compared to the observations made by a human analyst.

A user will be observed while performing a simple task. The system will run through its entire algorithm and attempt to gather performance data. This logged data will then be compared to the data gathered by the human analyst on the following points:

- **Accuracy of results** – compare the actual times for each work cycle
- **Accuracy of recognition** – compare the number of recognized cycles
- **The total training time** – the time taken for system to start gathering data

4.2.1. Setup

A worker was required to perform a simple task which emulates an assembly task, by retrieving Lego bricks from bins located on the workbench, assembling the bricks in a specific manner, and, and placing the completed assembly in a bin to the right hand side of the worker. The Kinect camera was placed on a tripod approximately 2.5 meters away from the worker, at a height of 1.8 meters. A laptop, which served as the execution platform for the program as well as the feedback mechanism to the worker, was placed to the left of the worker. Concurrently, a human analyst observing was given a stop-watch timer, and was told to note the time of each completed assembly. The worker, as seen by the camera, is shown in Figure 19 below.

.

Figure 19: Test setup showing worker and assembly bins with Lego bricks

4.2.2. Execution

1. The operator ensures the work scene is set up adequately and executes the program. The Kinect camera is adjusted so that the worker and all relevant objects in the scene are fully visible and not obscured.
2. The worker performs a calibration pose and waits for confirmation from the operator that she is being tracked.
3. The operator signals for the worker to start working and initiates the training process.
4. The worker performs the assembly task, placing completed components into the bin on the left. The operator observes and notes the time of each completed component.
5. 30 – 50 components are assembled. The operator ends the experiment by exiting the program.
6. Logged times from the log file are exported to excel for data analysis.

4.2.3. Data

Data was logged by the system as well as by the analyst. The times noted were compared using Microsoft Excel. A summary of the data is presented in 4.2.4, and the full logs are included in the appendix.

4.2.4. Results

This section will analyse the results obtained in section 4 and will comment on the effectiveness of the system as a whole, and the success of this project. Strong and weak points will be discussed.

4.2.4.1. Experiments Performed

Two experiments were performed according to the process prescribed in section 4.2.2. Experiment 1 was performed by a worker who was required to assemble 2 different Lego bricks using only her right hand. Experiment 2 allowed the worker to use both hands. An operator was present throughout the work performed, and used a digital stopwatch with an interval timer to manually record the completed work-cycles. The operator was not allowed to advise or interfere with the worker in any way, in order to simulate a real-world situation where the worker would be left unattended.

Using this method of experimentation, the accuracy of the system's timings could be tested, its ability to reliably count cycles, and it's flexibility to recognise different actions. The time taken and number of cycles taken to recognise a cycle was also recorded. To be noted is that a minimum of 3 cycles is required by the system for training purposes, and the first result which is timed is at least the 4^{th} cycle.

Table 1 and Table 2 below provide a summary of the results obtained from the two experiments.

Experiment no. 1			
Description:	Assemble 2 Lego bricks using right hand only		
	Operator	Kinect System	
Number of Events Recorded	50	45	
Number of Training Cycles Required	-	3	
Number of Failed Recognitions	-	2	
% Recognition Excluding Training	-	95.6%	
Average Time Recorded	8.08 sec	8.09 sec	
Time Taken to Train	-	25.98 sec	

Table 1: Comparative summary of results obtained Experiment 1

Experiment no. 2			
Description:	Assemble 2 Lego bricks using both hands		
	Operator	Kinect System	
Number of Events Recorded	60	57	
Number of Training Cycles Required	-	3	
Number of Failed Recognitions	-	0	
% Recognition Excluding Training	-	100.0%	
Average Time Recorded	5.08 sec	5.00 sec	
Time Taken to Train	-	20.39 sec	

Table 2: Comparative summary of results obtained Experiment 2

4.2.5. Discussion of Results

The results from the simulated experiment in section 4 show that under test circumstances, the system is a valid tool to perform time-study. Both experiments yielded favourable results, and the accuracy of the timing observed was very accurate. The time taken to train and the number of training cycles needed was deemed acceptable for both experiments.

Experiment 1 failed to recognise two cycles out of the 45 cycles which the operator observed. Failed tasks are easily filtered out of the data by noting that the times for failed recognition are

always approximately twice the length of the average of other observed tasks, and can thus be easily discarded by the analyst prior to further calculation or consideration of the data.

As can be seen in Experiment 2 recognized all cycles and had the minimum number of training cycles, and was thus ideal. The accuracy of the results obtained can be seen in Figure 20 below.

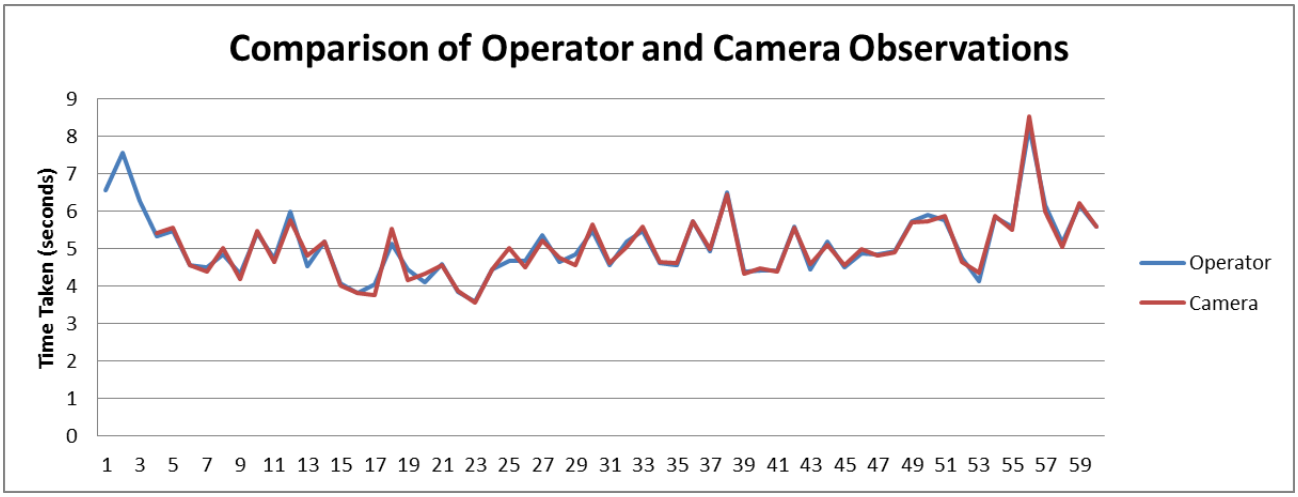


Figure 20: Comparison of Operator and Camera Observations

4.2.5.1. Problems Identified

The system often had trouble keeping a rigid track of the worker's hands, and the failed cycles observed in experiment 2 were as a result of this. The accuracy of the skeletal tracking was also noted to be less reliable when the subject handled objects or came into contact with the surface of the table. It was noticed that accuracy improved when more of the subject's body was visible.

5. Conclusion

Of all the alternative methods for performing cycle recognition, the one chosen was perhaps the least refined, and lacked the flexibility and robustness which some of the other methods could have provided. The main reason for its use was ease of implementation from a programming perspective – any of the other methods mentioned would have taken more time than the allocated project time to implement. The method, though, still performed very well when care was taken by both the operator and the worker, to ensure that the system had been trained to recognize a cycle properly. This intrusion into the worker's work and lack of intuitive and consistent operation was the biggest crux which could not be fully overcome.

The system was also at times limited by the hardware and software used. Although the Kinect camera and OpenNI framework are technologically an excellent system, they are not designed specifically for the uses as described in this project. A specific limitation was the accuracy of skeletal-tracking data when the worker's body was partially obscured or when the worker was handling objects. The Kinect occasionally struggled to keep an accurate lock on the user in these situations, and this was especially detrimental during the training of the system, which, due to the method of cycle recognition used, was sensitive to such behaviour.

6. Future Work

There are a number of improvements which can be made to the programming of the system. Even if no great changes are made, there is a lot of room to refine the program as there are many variables and methods which interact and cannot be optimized without extensive testing.

Future work should be focussed primarily on making the system more robust for cycle detection, in order to decrease the footprint and the intrusiveness of the system. Other methods of cycle detection should be studied, particularly those such as Hidden Markov Models and frequency domain analysis, which use probabilistic instead of binary detection methods and are thus likely to be more stable.

As the limitations of the hardware used are out of the control of this project and no alternatives exist which could provide better results overall, there is little that can be said about improving this aspect. However, due to the nature of technology in general it can only be foreseen than in the future devices similar to the Kinect will exist which will be more functional, less expensive, and smaller. The onus is still upon industrial engineers to make use of this technology in a beneficial way though.

A user interface for the operator should be implemented, though this is not seen as a priority.

As the Kinect is capable of tracking any number of people, as long as they fit within the camera's viewing field, the program could be extended to track multiple workers at the same time, thus reducing the cost of hardware.

The Kinect contains a microphone array, which could in future implementations serve as an intuitive way for the worker to interact with the system. Voice commands could be used by the worker to start and stop tracking, recalibrate the sensor, restart the training process, or even to make notes while working which an analyst could study at a later stage.

An extension to cycle recognition is action recognition. Action recognition could be used to identify the process a worker is performing, and automatically match the process to the correct process from a local database. The system could then provide feedback to the worker such as

training information, and may suggest improvements to the worker by comparing his actions to the actions of his peers, if his peers are performing better.

Analysts could walk around on a factory floor with handheld devices such as smartphones or tablets with attached 3D cameras, point the device at the worker, and immediately have all information on that task, such as rate of work, standard time, batch size, number completed, estimated time to completion, etc. Drivers for Android smartphone OS already been created, and the minimum specifications for the running of OpenNI are low enough for high end smartphones to run these applications, so this vision may not be too distant in the future.

REFERENCES

1. Asus. (2011). *Asus WAVI Xtion Pro: FAQ*. Retrieved September 26, 2011, from Asus WAVI Xtion Pro: http://event.asus.com/wavi/gallery/FAQ.aspx

2. Dreijer, J. (2011). Correspondence and Interview.

3. Dreijer, J. H. (2010). Action Classification using the Average of Pose Changes. *Pattern Recognition Association of South Africa* , 85-87.

4. Freivalds, A. (2009). *Niebel's Methods, Standards, and Work Design - International Edition.* McGraw Hill.

5. Microsoft Research. (2011, June). *About the Kinect for Windows SDK*. Retrieved October 24, 2011, from http://research.microsoft.com/en-us/um/redmond/projects/kinectsdk/about.aspx

6. Microsoft Research Cambridge & Xbox Incubation. (2011). Real-Time Human Pose Recognition in Parts from Single Depth Images.

7. Nissen, S. (2009). *Neural Networks Made Simple.* Retrieved June 06, 2011, from FANN: http://fann.sourceforge.net/fann_en.pdf

8. OpenNI Organization. (2011). Introducing OpenNI.

9. PrimeSense. (2011). *FAQ*. Retrieved June 13, 2011, from PrimeSense.com: http://www.primesense.com/?p=535

10. PrimeSense. (2011). *Reference Design*. Retrieved June 13, 2011, from PrimeSense.com: http://www.primesense.com/?p=514

11. Van Blommestein, D. (2010). *An Effective Labour Performance Measurement Aimed at Optimising Productivity for A South African Company.* Stellenbosch University.

12. Wassner, H. (2011, May 5). *Kinect + Reseau de Neurone = Reconnaissance de Gestes*. Retrieved June 17, 2011, from Hubert Wassner Professeur d'Informatique - Blog:

http://professeurs.esiea.fr/wassner/?2011/05/06/325-kinect-reseau-de-neurone-reconnaissance-de-gestes

13. Winston, W. (2004). *Operations Research Applications and Algorithms, Fourth Edition.* Brooks/Cole.

14. Wired.com. (2011). *How Motion Detection Works in the Xbox Kinect*. Retrieved June 13, 2011, from http://www.wired.com/gadgetlab/2010/11/tonights-release-xbox-kinect-how-does-it-work/

Appendix A: OpenNI and NITE Installation Instructions

Instructions obtained from: `http://entreprene.us/2011/03/21/how-to-set-up-openni-and-nite-on-windows/`

Phase 1

- Download **Kinect Drivers** and unzip the folder.
- Open the unzipped repository and move to Platform\Win32\Driver.
- Run **dpinst-x86.exe** (if you have a 32-bit processor) or **dpinst-amd64.exe** (if you have a 64-bit processor).

Drivers are now installed in your PC.

Phase 2

Download and install the latest stable or unstable **OpenNI Binaries** from OpenNI website.

Phase 3

Download and install the latest stable or unstable **OpenNI Compliant Middleware Binaries** (NITE) from OpenNI website. The installation process will ask for a product key. Not to worry, enter the following key to it: **0KOIk2JeIBYClPWVnMoRKn5cdY4=**

Phase 4

Download and install the latest stable or unstable **OpenNI Compliant Hardware Binaries** from OpenNI website.

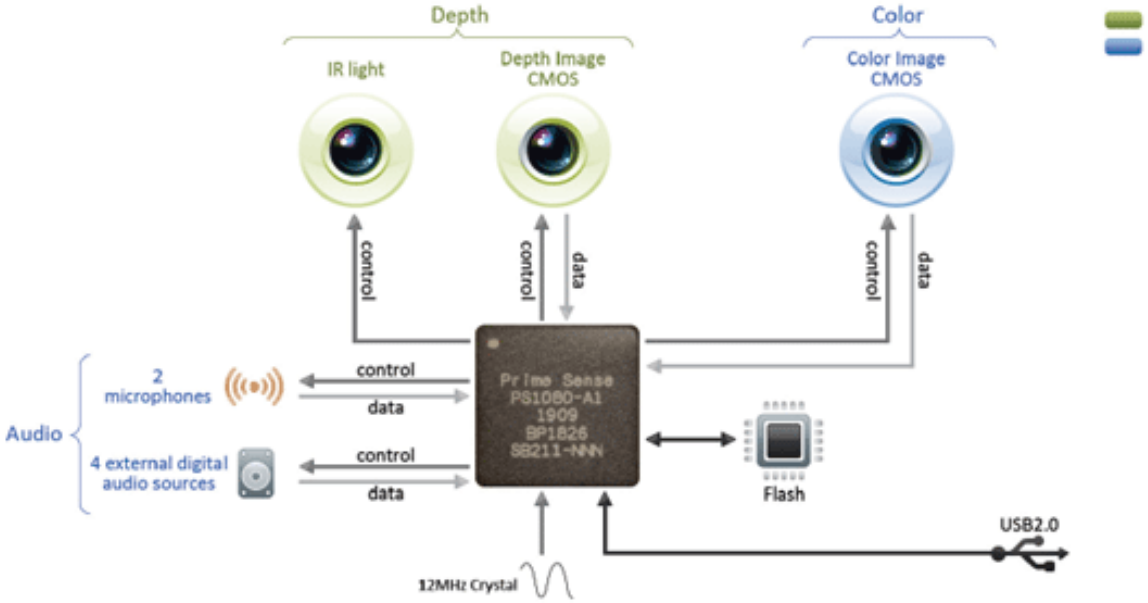
Phase 5

- Plug in your Kinect device and connect its USB port with your PC.
- Wait until the driver software is found and properly installed.
- Open your **Device Manager** from the Control Panel. Under *Other devices* you should see *Xbox NUI Audio*

Phase 7

Go to **C:\Program Files\OpenNI\Samples\Bin\Release** (or C:\Program Files (x86)\OpenNI\Samples\Bin\Release) and try out the existing demo applications. Also try the demos found in **C:\Program Files\Prime Sense\NITE\Samples\Bin\Release** (or C:\Program Files (x86)\Prime Sense\NITE\Samples\Bin\Release), too. If they work properly, then you are done! Congratulations!

Appendix B: Detailed Kinect Specifications



Specification

Property	Spec
Field of View (Horizontal, Vertical, Diagonal)	58° H, 45° V, 70° D
Depth image size	VGA (640x480)
Spatial x/y resolution (@ 2m distance from sensor)	3mm
Depth z resolution (@ 2m distance from sensor)	1cm
Maximum image throughput (frame rate)	60fps
Operation range	0.8m - 3.5m
Color image size	UXGA (1600x1200)
Audio: built-in microphones	Two mics
Audio: digital inputs	Four inputs
Data interface	USB 2.0
Power supply	USB 2.0
Power consumption	2.25W
Dimensions (Width x Height x Depth)	14cm x 3.5cm x 5cm
Operation environment (every lighting condition)	Indoor
Operating temperature	0°C - 40°C

Appendix C: File: *"CycleRecognition.cpp"*

```
#include "SceneDrawer.h"
#include <math.h>
#include <string.h>
#include <regex>
#include <iostream>
#include "CycleRecognition.h"
#include "SkeletonFunctions.h"
#include "StringFunctions.h"
#include <GL/glut.h>

#define ZONE_SIZE 4

using namespace std;

extern xn::UserGenerator g_UserGenerator;
extern xn::DepthGenerator g_DepthGenerator;

float field[81][61];
h_point points[20];

int currentState = 0;
int len;
bool h_bCR_Init = FALSE;
bool isTrained = FALSE;
int evtcount = 0;
int currentevent = 0;

FILE * outputFile;

float maxval = 0;

int t0;

int framecount = 0;

int prevy = 0;
int prevx = 0;

int ndp = 0;

float average;
float performance;
float ptime;

string evts;
string action[10];


int H_init()
{
	//LoadCalibration();

	if(g_UserGenerator.GetSkeletonCap().IsTracking(XnUserID(1)))
	{
		printf("Player is being tracked...\n");
```

```
		h_bCR_Init = TRUE;
		return 1;
	}
	else
	{
		printf("\rWaiting for user..");
		h_bCR_Init = FALSE;
		return 0;
	}
}

int EventCheck(int pointcount, h_point *points, XnPoint3D h_pt)
{
	int ReturnVal = 0;
	for (int i = 1; i <= pointcount; i++)
	{
		if (abs(points[i].X - h_pt.X) <= ZONE_SIZE && abs(points[i].Y - h_pt.Y) <=
ZONE_SIZE)
		{
			ReturnVal = i;
		}
	}
	return ReturnVal;
}

void GetSkeletonData(XnPoint3D *h_pt)
{
	if (!g_UserGenerator.GetSkeletonCap().IsTracking(XnUserID(1)))
	{
		printf("\rnot tracked!");
		return;
	}

	XnSkeletonJointPosition h_handPos;
	g_UserGenerator.GetSkeletonCap().GetSkeletonJointPosition(XnUserID(1),
XN_SKEL_RIGHT_HAND, h_handPos);

	if (h_handPos.fConfidence < 0.5)
	{
		return;
	}

	*h_pt = h_handPos.position;
	return;
}

void calculateFieldValues(int dt, XnPoint3D h_pt)
{
	float n;
	int px = (int) h_pt.X;
	int py = (int) h_pt.Y;

	n = (float) 20/50;

	if(currentevent != 0)
	{
		if(maxval*0.95 > points[currentevent].value)
		{
			n = n*2;
```

```
                }
        }

        if (abs(py-prevy)<=1 && abs(px-prevx)<=1)
        {
                n = n*2; //could be n*3 possibly?
        }

        for (int lx = px-2; lx <= px+2; lx++)
        {
                for (int ly = py-2; ly <= py+2; ly++)
                {
                        if (abs(ly-py) <= 2 && abs(lx-px) <= 2)
                        {
                                field[lx][ly] += n;
                        }
                        if (abs(ly-py) <= 1 && abs(lx-px) <= 1)
                        {
                                field[lx][ly] += n;
                        }
                        if (ly == py  && lx == px)
                        {
                                field[lx][ly] += 2*n;
                        }
                }
        }
        prevx = px;
        prevy = py;
}

void DrawFieldMap()
{
        float red;
        float green;
        bool alpha = 0;

        for (int i = 1 + ZONE_SIZE; i <= 80 - ZONE_SIZE; i++)
        {
                for(int j = 1 + ZONE_SIZE; j <= 60 - ZONE_SIZE; j++)
                {
                        if(maxval < field[i][j])
                        {
                                maxval = field[i][j];
                        }

                        if(field[i][j] > maxval*0.1f)
                        {
                                alpha = 1;
                                calculateColor(maxval, field[i][j], &red, &green);
                                glColor4f(red, green, 0, alpha);
                                DrawRectangle(i*8, j*8, i*8+4, j*8+4);
                        }
                        if (field[i][j] > 1 && !isTrained)
                        {
                                field[i][j] = field[i][j] - 0.1;
                        }
                }
        }
}
```

```
void findHighestPoints()
{
	int stop = 0;
	int pointcount = 0;
	for (int pp = 0; pp <= 20; pp++) //clear points -- does this do anything?
	{
		points[pp] = points[0];
	}
	for (int i = 1 + ZONE_SIZE; i < 80 - ZONE_SIZE; i++)
	{
		for (int j = 1 + ZONE_SIZE; j < 60 - ZONE_SIZE; j++)
		{
			stop = 0;
			if(field[i][j] > maxval*0.25f)
			{
				for (int li = i - ZONE_SIZE; li <= i + ZONE_SIZE; li++)
				{
					for (int lj = j - ZONE_SIZE; lj <= j + ZONE_SIZE; lj++)
					{
						for (int p = 1; p <= ndp; p++)
						{
							if(li > points[p].X - ZONE_SIZE && li <
points[p].X + ZONE_SIZE
								&& lj > points[p].Y - ZONE_SIZE &&
lj < points[p].Y + ZONE_SIZE)
							{
								stop++;
								p = ndp + 1;
							}
						}
						if(!(li == i && lj == j))
						{
							if(field[i][j] <= field[li][lj])
							{
								stop++;
								li = i + ZONE_SIZE + 1;
								lj = j + ZONE_SIZE + 1;
							}
						}
					}
				}
			}
			else{stop++;}
			if (stop == 0)
			{
				points[++pointcount].X = i;
				points[pointcount].Y = j;
				points[pointcount].value = field[i][j];
			}
		}
	}
	ndp = pointcount; //need a global (ndp) because not checking every frame
}

void H_CycleRecognition()
{
	if(!h_bCR_Init)
	{
```

```
		if(H_init())
		{
			printf("\nTraining started...");
		}
		else
		{
			return;
		}
	}

	int t1;
	int dt;

	t1 = GetTickCount();
	dt = t1 - t0;
	t0 = t1;

	XnPoint3D h_pt;
	GetSkeletonData(&h_pt);
	g_DepthGenerator.ConvertRealWorldToProjective(1, &h_pt, &h_pt);  //convert data to
projective and lower resolution
	H_ConvertRes(&h_pt, &h_pt, 8);

	if (!isTrained)
	{
		calculateFieldValues(dt, h_pt);
		findHighestPoints();
		DrawFieldMap();
	}

	displayPoints(ndp, ZONE_SIZE, points, isTrained);


	int t_currentevent;
	t_currentevent = EventCheck(ndp, points, h_pt);

	if(t_currentevent > 0 && t_currentevent != currentevent && ndp > 1)
	{
		currentevent = t_currentevent;
		evts.push_back(getCh(currentevent+64));
		std::string match;
		match = regexMatch(evts);

		if (match != "NO_MATCH")
		{
			bool foundInActionArray = FALSE;
			int i = 0;
			while(action[i] != "" && i < 10)
			{
				cout << action[i] << endl;
				if(action[i] == match)
				{
					cout << "found  " << match << " at " << i <<  endl;
					foundInActionArray = TRUE;
					break;
				}
				i++;
			}
			if(!foundInActionArray)
```

```
                        {
                                action[i] = match;
                                cout << i << "- ACTION ADDED:  " << action[i] << endl;
                                cout << "\nlist of actions held:"  << endl;
                                for(int j = 0; j <= i ; j++)
                                {
                                        cout << action[j] << endl;
                                }
                                isTrained = TRUE;

                        }
                }
                int i = 0;
                while(action[i] != "")
                {
                        if(FindSubstring(action[i], evts))
                        {
                                static int etprev;

                                cout << "ACTION FOUND:  " << action[i] << endl;
                                evtcount++;
                                evts.clear();

                                //HERE COMES THE PERFORMANCE CALCULATION!
                                float etnow = GetTickCount();
                                ptime = etnow - etprev;
                                        etprev = etnow;

                                if (evtcount > 1) //we discard the first event!
                                {
                                        evtcount--;
                                        average = (average*(evtcount-1) + ptime)/evtcount;
                                        performance = (float) (average/ptime);
                                        //cout << num << endl;
                                        evtcount++;

                                //OUTPUT TO FILE
                                time_t rawtime;
                                struct tm * timeinfo;
                                time ( &rawtime );
                                timeinfo = localtime ( &rawtime );

                                outputFile = fopen ("tracking_output.txt","a");
                                fprintf(outputFile, "%f\t%s",  ptime, asctime (timeinfo));
                                fclose(outputFile);
                                }
                        }
                        i++;
                }
        }
        if ( evtcount > 1)
                {
                        displayPerformanceIndicator(performance, evtcount, ptime/1000,
average/1000);
                }
}
```

Appendix D: File: "*StringFunctions.cpp*"

```
#include "StringFunctions.h"

char getCh(int Ch)
{
	char returnChar;
	returnChar = Ch;
	return returnChar;
}

std::string regexMatch(const std::string& evt)
{
// regular expression
	const std::tr1::regex pattern("(.{3,}?)\\1{2,}");

	// sequence of string sub-matches
	std::tr1::smatch result;

	bool match = std::tr1::regex_search(evt, result, pattern);

	if(match)
	{
		// if there was a match print it
		return result[result.size()-1];
	}
	else {return "NO_MATCH";}

}

int FindSubstring(std::string& action, std::string& evts) //this works
{
	//bool go = 1;
	int n = 0;


	for (int i = 0; i < action.length(); i++)
	{
		while (true)
		{
			if(n == evts.length()) //if(n = evts.length()) >>>null terminated??
			{
				//go = 0;
				return 0;
			}
			if(evts[n] == action[i])
			{
				n++;
				break;
			}
			n++;
		}
	}
	//printf("FOUND substring %s AT:%d\n", action, n);
	return 1;
}
```

Appendix E: File: "*SkeletonFunctions.cpp*"

```
#include <XnCppWrapper.h>
#include <GL/glut.h>
#include "StringFunctions.h"
#include "SkeletonFunctions.h"
#include "SceneDrawer.h"
#include <string.h>
#include <string>
#include <sstream>
using namespace std;


void glPrintString_H(void *font, char *str)
{
	int i,l = strlen(str);

	for(i=0; i<l; i++)
	{
		glutBitmapCharacter(font,*str++);
	}
}

void H_ConvertRes(XnPoint3D *oldPt, XnPoint3D *newPt, int factor)
{
	newPt->X = (int) (oldPt->X)/factor;
	if(newPt->X > 80){newPt->X = 81;}
	if(newPt->X < 1){newPt->X = 0;}

	newPt->Y = (int) (oldPt->Y)/factor;
	if(newPt->Y > 60){newPt->Y = 61;}
	if(newPt->Y < 1){newPt->Y = 0;}

	newPt->Z = (int) (oldPt->Z)/factor;
	if(newPt->Z > 60){newPt->Z = 61;}
	if(newPt->Z < 1){newPt->Z = 0;}

	return;
}

void calculateColor(float max, float cur, float *red, float *green)
{
	*red = cur/max;
	*green = 1 - cur/max;
	return;
}

void DrawRectangleOutline(float topLeftX, float topLeftY, float bottomRightX, float
bottomRightY)
{
	GLfloat verts[8] = { topLeftX, topLeftY,
		topLeftX, bottomRightY,
		bottomRightX, bottomRightY,
		bottomRightX, topLeftY
	};
	glVertexPointer(2, GL_FLOAT, 0, verts);
	glDrawArrays(GL_LINE_LOOP, 0, 4);
```

```
	glFlush();
}

void displayPoints(int ndp, int zone_size, h_point *points, bool isTrained)
{
	for(int l = 1; l <= ndp; l++)
	{
		char disp = getCh(l + 64);
		if(isTrained)
		{
			glRasterPos2i(points[l].X*8, points[l].Y*8);
			glPrintString_H(GLUT_BITMAP_HELVETICA_18, &disp);
		}
		DrawRectangleOutline(8*(points[l].X - zone_size), 8*(points[l].Y -
zone_size), 8*(points[l].X + zone_size), 8*(points[l].Y + zone_size));
	}
}

void calcFeedbackColor(float per, float *red, float *green)
{
	float mid = 0.85;
	float low = 0.70;

	if(per >= 1)
	{
		*red = 0;
		*green =1;
	}
	else if(per >= mid)
	{
		*red = 1 - (per - mid)/(1 - mid);
		*green = 1;
	}
	else if (per >= low)
	{
		*red = 1;
		*green = (per - low)/(mid - low);
	}
	else
	{
		*red = 1;
		*green =0;
	}
}

void displayPerformanceIndicator(float per, int evtcount, float ptime, float average)
{
	float red, green;
	char info[200];
	char output[10];

	//RECTANGLES AT BOTTOM & SIDES
	calcFeedbackColor(per, &red, &green);
	glColor4f(red, green, 0, 1);
	DrawRectangle(0, 480 - 64, 640, 480);
	DrawRectangle(0, 24, 24, 480 - 60);
	DrawRectangle(640 - 24, 24, 640, 480 - 24);
```

```
       _snprintf(output, 8, "%5.2f%c", per*100, '%');

    glColor4f(0.212, 0.224, 0.217, 1); //BM black knight (dark grey)
       glRasterPos2i(320 - 3, 480 - 18 - 7 );
       glPrintString(GLUT_BITMAP_HELVETICA_18, output);

       //STATUS BAR AT THE TOP
       DrawRectangle(0,0, 640, 24);


       _snprintf(info, 198, "Last Action: %5.2fsec    Average: %5.2fsec    Events
Counted: %2d", ptime, average, evtcount+2);

       glColor4f(0.855, 0.847, 0.8, 1); //BM winter orchard (off-white)
       glRasterPos2i(8,18);
       glPrintString(GLUT_BITMAP_HELVETICA_18, info);
}
```